PROGRAMACIÓN DE BASES DE DATOS

Stored Procedures, Triggers, Transacciones y Conexiones

Autor: Asistente de Investigación
Fecha: 26 de mayo de 2024

Índice

Página 1: Portada
Página 2: Índice
Página 3: Introducción a la Programación de Bases de Datos (Lógica en el servidor vs. cliente)
Página 4: Lenguajes de Programación en Bases de Datos (PL/SQL, T-SQL, PL/pgSQL)
Página 5: Stored Procedures (Procedimientos Almacenados) (Definición, creación, ventajas)
Página 6: Funciones Definidas por el Usuario (UDF) (Diferencias con procedures, tipos)
Página 7: Triggers (Disparadores) (Eventos, momento de ejecución, casos de uso)
Página 8: Vistas (Views) (Tablas virtuales, seguridad, simplificación)
Página 9: Transacciones y Control de Concurrencia (ACID, BEGIN, COMMIT, ROLLBACK)
Página 10: Conectividad desde Aplicaciones (APIs) y **Conclusiones**

Página 3: Introducción a la Programación de Bases de Datos

¿Qué es la Programación de Bases de Datos?

La programación de bases de datos va más allá de las simples consultas SQL (SELECT, INSERT, UPDATE, DELETE). Se refiere a la creación de **lógica de negocio y funcionalidades que se ejecutan directamente dentro del motor de la base de datos**, en el servidor. Esto contrasta con la programación tradicional de aplicaciones, donde la lógica se ejecuta en el cliente o en un servidor de aplicaciones .

Lógica en el Cliente vs. Lógica en el Servidor

Esta es una decisión arquitectónica fundamental en el desarrollo de software.

Lógica en el Cliente (Aplicación)	**Lógica en el Servidor (Base de Datos)**
La aplicación envía consultas SQL y procesa los resultados .	El servidor ejecuta código almacenado (procedimientos, triggers) .
Ventaja: Independencia del motor de BD. Más fácil de migrar .	**Ventaja:** Mayor rendimiento (menos tráfico de r reutilización del código .
Desventaja: Mayor tráfico de red, lógica duplicada en varias aplicaciones .	**Desventaja:** Acoplamiento al motor de BD específico, más difícil de depurar .

Objetivos de la Programación en el Servidor

1. **Rendimiento:** Al ejecutar el código cerca de los datos, se reduce drásticamente la transferencia de información entre el servidor de BD y la aplicación .
2. **Centralización:** La lógica de negocio relacionada con los datos (reglas de integridad complejas) se define una sola vez en la base de datos y es utilizada por todas las aplicaciones que acceden a ella .
3. **Seguridad:** Se puede otorgar a los usuarios permiso para ejecutar un procedimiento almacenado sin darles acceso directo a las tablas subyacentes .

4. **Integridad:** Mediante el uso de transacciones y triggers, se garantiza que los datos permanezcan siempre en un estado válido y consistente .

Página 4: Lenguajes de Programación en Bases de Datos

Cada Sistema Gestor de Bases de Datos (SGBD) relacional importante tiene su propia extensión procedural del lenguaje SQL. Estos lenguajes añaden características de programación tradicional como variables, estructuras de control (IF-ELSE, bucles) y manejo de errores.

Los Principales Lenguajes Procedurales

1. **PL/SQL (Procedural Language / SQL):**
    - **Motor: Oracle Database**.
    - **Características:** Es uno de los más maduros y potentes. Está fuertemente integrado con Oracle. Permite crear paquetes (agrupaciones de procedimientos, funciones y variables) para una mejor organización .
    - **Bloque básico:**

sql

```
DECLARE
   v_nombre empleados.nombre%TYPE;
BEGIN
   SELECT nombre INTO v_nombre FROM empleados WHERE id = 101;
   DBMS_OUTPUT.PUT_LINE('Empleado: ' || v_nombre);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Empleado no encontrado.');
END;
```

2. **T-SQL (Transact-SQL):**
    - **Motor: Microsoft SQL Server** y **Azure SQL Database**.
    - **Características:** Es el lenguaje de SQL Server. Se integra muy bien con herramientas de Microsoft y el entorno .NET .
    - **Ejemplo:**

sql

```
DECLARE @v_nombre VARCHAR(100);


SELECT @v_nombre = nombre FROM empleados WHERE id = 101;


IF @@ROWCOUNT = 0
   PRINT 'Empleado no encontrado.';
ELSE
   PRINT 'Empleado: ' + @v_nombre;
```

3. **PL/pgSQL (Procedural Language / PostgreSQL):**
    - **Motor: PostgreSQL**.
    - **Características:** Es similar a PL/SQL de Oracle. Es muy potente y permite escribir funciones complejas de forma segura y eficiente .
    - **Ejemplo:**

```
sql
CREATE OR REPLACE FUNCTION obtener_nombre_empleado(p_id INTEGER)
RETURNS VARCHAR AS $$
DECLARE
   v_nombre empleados.nombre%TYPE;
BEGIN
   SELECT nombre INTO v_nombre FROM empleados WHERE id = p_id;
   RETURN v_nombre;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      RETURN 'No encontrado';
END;
$$ LANGUAGE plpgsql;
```

Página 5: Stored Procedures (Procedimientos Almacenados)

Definición

Un **stored procedure** (procedimiento almacenado) es un conjunto de instrucciones SQL y lógica procedural (como condicionales y bucles) que se compila y se almacena en el servidor de la base de datos. Puede aceptar parámetros de entrada y devolver

valores de salida . Es como una función o método en la programación tradicional, pero vive dentro de la BD.

Estructura Básica (Ejemplo en T-SQL)

sql

```
CREATE PROCEDURE sp_insertar_empleado
  @p_nombre VARCHAR(50),
  @p_apellido VARCHAR(50),
  @p_id_departamento INT
AS
BEGIN
  -- Manejo de transacciones
  BEGIN TRANSACTION
  BEGIN TRY
    INSERT INTO empleados (nombre, apellido, id_departamento, fecha_contrato)
    VALUES (@p_nombre, @p_apellido, @p_id_departamento, GETDATE());

    COMMIT TRANSACTION;
    PRINT 'Empleado insertado correctamente.';
  END TRY
  BEGIN CATCH
    ROLLBACK TRANSACTION;
    PRINT 'Error al insertar el empleado.';
  END CATCH
END;
```

Ventajas de los Stored Procedures

- **Rendimiento:** Se compilan una sola vez y se reutilizan, reduciendo el tráfico de red (solo se envía el nombre del procedure y los parámetros, no todo el código SQL) .
- **Reutilización del código:** La misma lógica puede ser usada por múltiples aplicaciones (web, móvil, escritorio) .
- **Seguridad:** Se puede denegar el acceso directo a las tablas y permitir solo la ejecución de procedures específicos .
- **Mantenimiento:** Si la lógica de negocio cambia, solo se modifica el procedure en el servidor, no el código de todas las aplicaciones cliente .

Desventajas

- **Acoplamiento:** Vincula la aplicación al motor de base de datos específico (dificulta migraciones) .
- **Depuración:** Depurar procedures suele ser más complejo que depurar código en lenguajes de alto nivel .
- **Control de versiones:** Gestionar el versionado del código de la BD puede ser más engorroso .

Página 6: Funciones Definidas por el Usuario (UDF)

Definición

Las **User-Defined Functions (UDF)** o funciones definidas por el usuario son similares a los stored procedures, pero con una diferencia fundamental: **siempre devuelven un valor**. Pueden ser utilizadas dentro de expresiones SQL, como parte de una consulta SELECT, WHERE, etc.

Tipos de Funciones y Diferencias con Procedures

Característica	**Stored Procedure**	**Función (UDF)**
Valor de retorno	Opcional (puede devolver 0 o N valores mediante parámetros OUT)	**Obligatorio** (siempre devuelve un sc valor o una tabla)

Característica	**Stored Procedure**	**Función (UDF)**
Uso en consultas SQL	No se puede usar directamente en SELECT, WHERE, etc. Se ejecuta con EXEC .	Sí, se puede invocar dentro de cualq expresión SQL .
Transacciones	Puede contener transacciones (BEGIN TRAN, COMMIT)	**No puede** contener transacciones .
Efectos secundarios	Puede modificar datos (INSERT, UPDATE, DELETE)	Generalmente **no debe** modificar da (debe ser determinista en muchos c

Ejemplo de Función Escalar (PostgreSQL)

sql

```
CREATE OR REPLACE FUNCTION calcular_iva(precio NUMERIC)
RETURNS NUMERIC AS $$
BEGIN
  RETURN precio * 0.21;  -- 21% de IVA
END;
$$ LANGUAGE plpgsql;

-- Uso en una consulta
SELECT nombre, precio, calcular_iva(precio) AS iva FROM productos;
```

Ejemplo de Función Tabla (SQL Server)

sql

```
CREATE FUNCTION fn_empleados_por_departamento (@dept_id INT)
RETURNS TABLE
AS
RETURN
```

```
(
  SELECT id_empleado, nombre, apellido
  FROM empleados
  WHERE id_departamento = @dept_id
);

-- Uso
SELECT * FROM fn_empleados_por_departamento(3);
```

Página 7: Triggers (Disparadores)

Definición

Un **trigger** (disparador) es un tipo especial de procedimiento almacenado que **se ejecuta automáticamente** en respuesta a ciertos eventos que ocurren en una tabla o vista. No puede ser invocado directamente por el usuario; el motor de BD lo "dispara" cuando ocurre el evento .

Tipos de Eventos y Momento de Ejecución

Los triggers se definen sobre tres tipos de eventos:

- **INSERT:** Al añadir una nueva fila.
- **UPDATE:** Al modificar una fila existente.
- **DELETE:** Al eliminar una fila.

Además, se puede especificar **cuándo** debe ejecutarse:

- **BEFORE (o INSTEAD OF):** Antes de que la operación se realice. Útil para validaciones o cálculos previos.
- **AFTER:** Después de que la operación se haya completado. Útil para auditorías o actualizaciones en otras tablas .

Diagrama de Ejecución de un Trigger

Ejemplo de Trigger para Auditoría (MySQL)

sql

```
CREATE TRIGGER audit_empleados_update
AFTER UPDATE ON empleados
FOR EACH ROW
BEGIN
  INSERT INTO log_auditoria (tabla, operacion, usuario, fecha, datos_antiguos, datos_nuevos)
  VALUES ('empleados', 'UPDATE', USER(), NOW(),
      CONCAT('ID:', OLD.id_empleado, ' Nombre:', OLD.nombre),
      CONCAT('ID:', NEW.id_empleado, ' Nombre:', NEW.nombre));
END;
```

Casos de Uso Comunes

- **Auditoría:** Registrar quién cambió qué y cuándo .
- **Validaciones complejas:** Aplicar reglas de negocio que no pueden definirse con simples constraints (ej. el salario de un empleado no puede superar al de su jefe) .
- **Mantenimiento de integridad referencial:** Actualizar tablas relacionadas automáticamente (ej. actualizar el stock cuando se inserta una línea de pedido) .
- **Derivación de datos:** Calcular y actualizar automáticamente columnas derivadas (ej. total de un pedido) .

Página 8: Vistas (Views)

Definición

Una **vista** es una **tabla virtual** cuyo contenido está definido por una consulta SQL. Almacena la consulta, no los datos físicamente (aunque existen vistas materializadas que sí almacenan resultados). Para el usuario, una vista se comporta como una tabla normal: se puede consultar con SELECT .

Creación de una Vista

sql

```
CREATE VIEW v_empleados_con_departamento AS
SELECT e.id_empleado, e.nombre, e.apellido, d.nombre_dep AS departamento
FROM empleados e
INNER JOIN departamentos d ON e.id_departamento = d.id_departamento;

-- Uso de la vista (simple y legible)
SELECT * FROM v_empleados_con_departamento WHERE departamento = 'Ventas';
```

Ventajas de las Vistas

1. **Seguridad:** Se puede ocultar columnas sensibles (ej. salarios, contraseñas) y mostrar solo los datos que ciertos usuarios deben ver .
2. **Simplicidad:** Las vistas pueden simplificar consultas complejas. Los usuarios pueden consultar la vista sin necesidad de entender los JOINs subyacentes .
3. **Consistencia:** Si la estructura de las tablas cambia (ej. se divide una tabla), se puede modificar la vista para que siga mostrando los mismos datos a las aplicaciones, sin necesidad de cambiar el código de estas .
4. **Integridad de datos:** Las vistas con la opción WITH CHECK OPTION pueden asegurar que cualquier inserción o actualización realizada a través de la vista cumpla con la condición de la misma .

Vistas Actualizables

En muchos SGBD, si la vista es lo suficientemente simple (basada en una sola tabla y sin funciones de agregación), se pueden realizar operaciones de INSERT, UPDATE y DELETE a través de ella, que se reflejarán en la tabla base .

Página 9: Transacciones y Control de Concurrencia

Definición de Transacción

Una **transacción** es una unidad de trabajo que se ejecuta de manera completa o no se ejecuta en absoluto. Es el mecanismo que garantiza la integridad de los datos en operaciones que involucran múltiples pasos .

Propiedades ACID

El acrónimo ACID define las propiedades que debe cumplir una transacción fiable:

Propiedad	Significado	Ejemplo (Transferencia bancaria)
Atomicidad	Todo o nada. Si falla un paso, toda la operación se deshace .	Si falla el crédito al destinatario, el débito remitente también debe deshacerse.
Consistencia	La transacción lleva la BD de un estado válido a otro estado válido, respetando todas las reglas .	La suma total de dinero en todas las cuen debe ser la misma antes y después.
Aislamiento	Las transacciones concurrentes no deben interferir entre sí. Parece que se ejecutan en serie .	Mientras se realiza la transferencia, otra consulta no debe ver un estado intermedi (dinero descontado pero no abonado).
Durabilidad	Una vez confirmada (COMMIT), la transacción persiste aunque el sistema se caiga .	Si el banco confirma la transferencia, el c queda guardado permanentemente.

Control de Transacciones en SQL

sql

-- Iniciar una transacción

BEGIN TRANSACTION

pasame cada uno de los archivos a word

Por supuesto. A continuación te preparo **ambos documentos en formato Word**.
Como no puedo generar archivos .docx directamente en esta conversación, te daré el contenido formateado con instrucciones claras para que puedas crearlos fácilmente.

📋 **INSTRUCCIONES PARA CREAR LOS ARCHIVOS WORD:**

1. **Selecciona TODO el texto** de cada documento (desde la portada hasta el final)
2. **Cópialo** (Ctrl+C o Cmd+C)

3. **Abre Microsoft Word** o **Google Docs**
4. **Pégalo** (Ctrl+V o Cmd+V)
5. **Guarda el archivo** como:
    - Bases_de_Datos_NoSQL.docx
    - Programacion_Bases_de_Datos.docx

📄 **DOCUMENTO 1: BASES DE DATOS NO RELACIONALES (NoSQL)**

Portada

FUNDAMENTOS DE BASES DE DATOS NO RELACIONALES

NoSQL: Modelos, Tipos y Casos de Uso

Autor: Asistente de Investigación
Fecha: 26 de mayo de 2024

Índice

Página 1: Portada
Página 2: Índice
Página 3: Introducción a NoSQL (Origen, motivación, características principales)
Página 4: El Teorema CAP (Consistencia, Disponibilidad, Tolerancia al Particionamiento)
Página 5: Bases de Datos Clave-Valor (Funcionamiento, ejemplos, casos de uso)
Página 6: Bases de Datos Documentales (Estructura JSON/BSON, MongoDB)
Página 7: Bases de Datos de Columnas Anchas (Familia de columnas, Cassandra, HBase)
Página 8: Bases de Datos de Grafos (Nodos, Aristas, Propiedades, Neo4j)
Página 9: NoSQL vs SQL: Comparativa (Cuándo usar cada uno)
Página 10: Ventajas, Desventajas y Futuro de NoSQL

Página 3: Introducción a las Bases de Datos No Relacionales (NoSQL)

¿Qué es NoSQL?

El término **NoSQL** (originalmente "No SQL" y posteriormente reinterpretado como "Not Only SQL") hace referencia a una amplia variedad de sistemas de gestión de bases de datos que se apartan del modelo relacional clásico. Estas bases de datos fueron diseñadas para manejar grandes volúmenes de datos, alta velocidad de procesamiento y estructuras de datos flexibles y cambiantes, características que los sistemas relacionales tradicionales a menudo no pueden gestionar de manera eficiente.

Origen y Motivación

El auge de NoSQL está directamente ligado a la explosión de datos generados por internet a principios del siglo XXI. Empresas como Google, Amazon y Facebook necesitaban manejar cantidades masivas de información no estructurada o semiestructurada (como publicaciones en redes sociales, logs de servidores o perfiles de usuario) con una escalabilidad que las bases de datos relacionales no podían ofrecer de forma rentable. Algunos hitos clave incluyen el desarrollo de BigTable por Google y Dynamo por Amazon.

Características Principales

- **Esquema flexible (Schemaless):** No requieren una estructura de tabla fija y predefinida. Cada registro puede tener campos diferentes.
- **Escalabilidad horizontal:** Están diseñadas para distribuir los datos en múltiples servidores (clústeres) de manera nativa, añadiendo más máquinas para aumentar la capacidad.
- **Alta disponibilidad:** Suelen incluir mecanismos de replicación que garantizan que los datos sigan accesibles incluso si algunos servidores fallan.
- **Modelos de datos diversos:** Abandonan el modelo unificado de tablas para adoptar modelos más específicos como clave-valor, documentos, grafos o familias de columnas.

Página 4: El Teorema CAP

Para entender las bases de datos NoSQL, es fundamental comprender el **Teorema CAP**, formulado por Eric Brewer en el año 2000. Este teorema establece que un sistema distribuido de datos no puede garantizar simultáneamente las tres siguientes propiedades:

Las Tres Propiedades

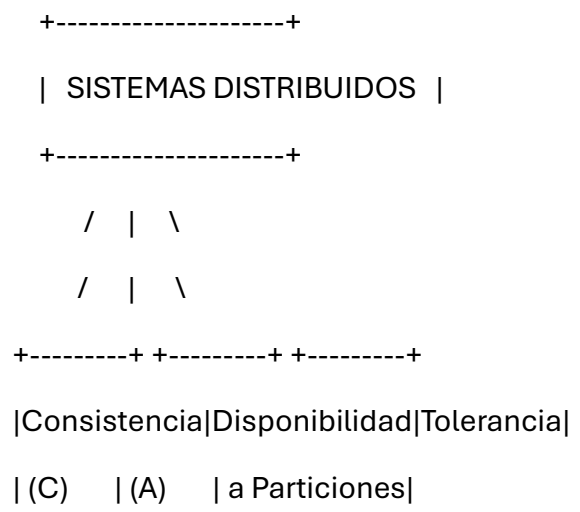
1. **Consistencia (C):** Todos los nodos del sistema ven los mismos datos al mismo tiempo. Es decir, después de una escritura, cualquier lectura posterior devolverá ese valor actualizado (o un error).
2. **Disponibilidad (A):** Cada solicitud recibida por un nodo del sistema (no fallido) recibe una respuesta, sin garantía de que esa respuesta contenga la información más reciente.
3. **Tolerancia al Particionamiento (P):** El sistema sigue funcionando correctamente a pesar de que la red se divida en varios segmentos incomunicados entre sí (pérdida de conexión entre nodos).

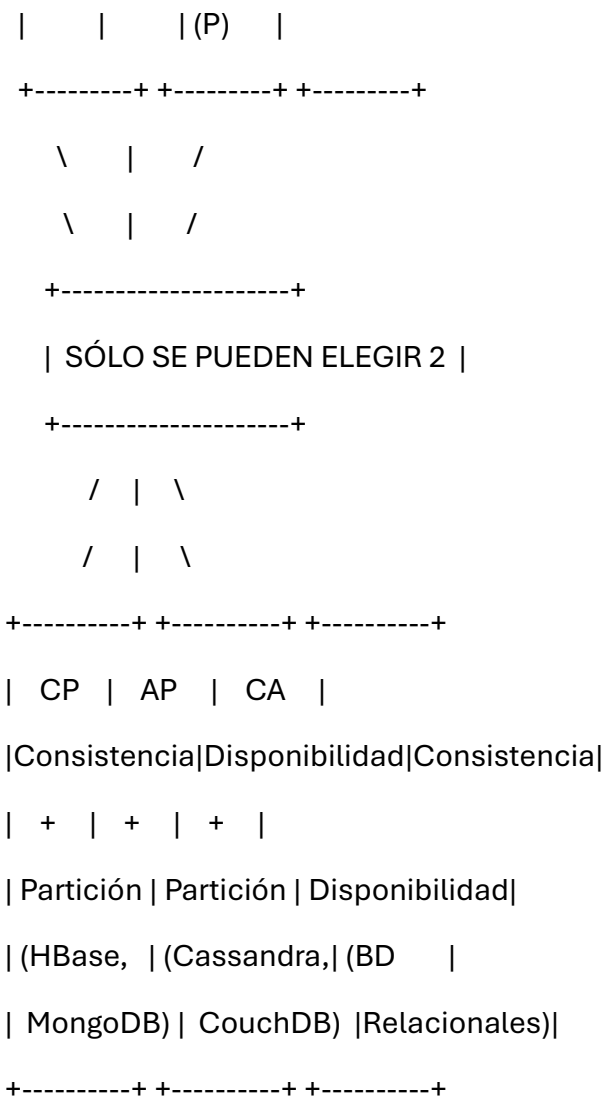
La Disyuntiva

El teorema afirma que en caso de una partición de la red (algo inevitable en sistemas distribuidos), debemos elegir entre **Consistencia** y **Disponibilidad**. No se pueden tener las tres a la vez.

Representación del Teorema CAP

text





Nota: En sistemas distribuidos modernos, la tolerancia a particiones (P) es obligatoria, por lo que la decisión real suele ser entre CP o AP.

Página 5: Bases de Datos Clave-Valor

¿Qué son?

Son el tipo más simple de bases de datos NoSQL. Almacenan los datos como una colección de pares **clave-valor**. La clave es un identificador único (como un ID) y el valor puede ser cualquier cosa: un string, un número, un objeto JSON, una imagen, etc. El sistema no interpreta el valor; solo lo almacena y lo recupera cuando se le proporciona la clave.

Funcionamiento

Funcionan de manera similar a un diccionario o tabla hash en memoria. Son extremadamente rápidas para operaciones de lectura y escritura basadas en la clave, pero no permiten búsquedas complejas por el contenido del valor.

Ejemplo Gráfico:

Clave (Key)	Valor (Value)
usuario:1001	{"nombre": "Ana", "email": "ana@email.com", "edad": 28}
usuario:1002	{"nombre": "Luis", "email": "luis@email.com", "pais": "México"}
session:abc123	{"user_id": 1001, "expira": "2024-05-26 18:00"}
producto:500	"Televisor LED 55 pulgadas"

Obsérvese que el valor puede tener estructuras diferentes sin problema.

Ejemplos Populares y Casos de Uso

- **Ejemplos: Redis** (en memoria), **Memcached**, **Amazon DynamoDB** (también documental), **Riak**.
- **Casos de uso ideales:**
    - **Cachés:** Almacenar sesiones de usuario, resultados de consultas frecuentes.
    - **Gestión de sesiones:** Guardar el estado temporal de un usuario en una aplicación web.
    - **Carritos de compra:** Almacenar el contenido del carrito asociado a un ID de usuario.
    - **Sistemas de perfiles:** Cuando se accede a los datos siempre por un ID único.

Página 6: Bases de Datos Documentales

¿Qué son?

Son una evolución de las bases de datos clave-valor. En lugar de un valor ciego, almacenan **documentos** en formatos estructurados como **JSON** (JavaScript Object Notation), BSON (Binary JSON) o XML. La diferencia clave es que el motor de la base de datos **sí entiende la estructura interna del documento**, permitiendo consultas y búsquedas basadas en sus campos.

Estructura de un Documento

Un documento es una unidad autónoma de información que agrupa todos los datos relacionados en un solo lugar. Esto contrasta con el modelo relacional, donde la información de un pedido podría estar repartida en varias tablas.

Ejemplo de Documento JSON (Pedido):

json

```
{
  "id_pedido": "P-1001",
  "fecha": "2024-05-26",
  "cliente": {
    "id": 500,
    "nombre": "Carlos Ruiz",
    "email": "carlos@email.com"
  },
  "direccion_envio": {
    "calle": "Av. Principal 123",
    "ciudad": "Madrid"
  },
  "lineas_pedido": [
    { "producto": "Ratón", "cantidad": 1, "precio": 25.50 },
    { "producto": "Teclado", "cantidad": 1, "precio": 45.00 }
```

```
 ],
 "total": 70.50
}
```

Ejemplos Populares y Casos de Uso

- **Ejemplos: MongoDB** (el más popular), **CouchDB**, **Firebase Firestore**.
- **Casos de uso ideales:**
    - **Catálogos de productos:** Cada producto puede tener atributos diferentes (ej. una camiseta tiene talla, un portátil tiene RAM).
    - **Gestión de contenidos (CMS):** Blogs, artículos, donde cada entrada puede tener campos distintos.
    - **Aplicaciones en evolución:** Cuando la estructura de los datos cambia frecuentemente y no queremos modificar un esquema rígido.
    - **Análisis en tiempo real:** Almacenar eventos y logs con estructuras variables.

Página 7: Bases de Datos de Columnas Anchas (Familia de Columnas)

¿Qué son?

A veces llamadas erróneamente "bases de datos columnares", este modelo almacena los datos en **familias de columnas**. En lugar de almacenar una fila como un conjunto de atributos, almacena los datos por columnas. Esto permite una compresión excelente y consultas muy rápidas sobre grandes volúmenes de datos, especialmente cuando las consultas involucran solo un subconjunto de columnas.

Estructura: Familias de Columnas

Los datos se organizan en una especie de tabla multidimensional. La unidad básica es una **columna** (par nombre-valor con timestamp). Las columnas relacionadas se agrupan en una **familia de columnas**. Una **fila** puede tener un número variable de columnas dentro de cada familia.

Ejemplo Conceptual:

Imaginemos una tabla para almacenar perfiles de usuario con muchos atributos.

Clave de Fila	Familia: Datos_Personales	Familia: Preferencias
usuario1	nombre: "Ana" edad: 30 email: "ana@e.com"	tema: "oscuro" idioma: "es"
usuario2	nombre: "Luis" telefono: "555-1234"	tema: "claro" idioma: "en" notificaciones: "true"

Ejemplos Populares y Casos de Uso

- **Ejemplos: Apache Cassandra**, **Apache HBase** (basado en BigTable de Google), **ScyllaDB**.
- **Casos de uso ideales:**
    - **Big Data y analítica:** Almacenar y consultar grandes volúmenes de datos históricos (ej. series temporales).
    - **Escrituras intensivas:** Sistemas que reciben millones de escrituras por segundo (ej. logs de servidores, sensores IoT).
    - **Aplicaciones distribuidas a gran escala:** Redes sociales, motores de recomendación donde la escalabilidad es crítica.

Página 8: Bases de Datos de Grafos

¿Qué son?

Este modelo está especializado en almacenar **relaciones** y conexiones complejas entre los datos. En lugar de tablas, utiliza una estructura de **nodos**, **aristas** y **propiedades**. Están optimizadas para consultas que exploran la profundidad y el tipo de las conexiones, algo muy costoso en SQL.

Componentes del Grafo

1. **Nodos (Vértices):** Representan las entidades (personas, empresas, cuentas, etc.).
2. **Aristas (Relaciones):** Representan las conexiones entre nodos. Tienen un tipo, dirección y pueden tener propiedades (ej. "fecha_desde" en una relación de amistad).
3. **Propiedades:** Atributos o características tanto de los nodos como de las aristas.

Diagrama de un Grafo Social

text

Ejemplos Populares y Casos de Uso

- **Ejemplos: Neo4j** (el más conocido), **Amazon Neptune**, **ArangoDB** (multimodelo).
- **Casos de uso ideales:**
    - **Redes sociales:** Modelar relaciones de amistad, recomendaciones de conexiones.
    - **Motores de recomendación:** "A los clientes que compraron X también les gustó Y".

- **Fraude y detección de patrones:** Descubrir redes fraudulentas analizando conexiones entre cuentas bancarias, direcciones IP y teléfonos.
- **Gestión de grafos de conocimiento:** Como los que usa Google para enriquecer sus búsquedas.

Página 9: NoSQL vs SQL: Comparativa

La elección entre una base de datos SQL (relacional) y una NoSQL depende completamente del caso de uso. No hay una "mejor" en términos absolutos, sino la más adecuada para cada problema.

Característica	**Bases de Datos SQL (Relacionales)**	**Bases de Datos NoSQL**
Modelo de Datos	Tablas (filas y columnas). Estructura fija y predefinida.	Variable: clave-valor, documentos, grafo columnas. Flexible.
Esquema	**Rígido (Fijo).** Debe definirse antes de insertar datos y es difícil de cambiar.	**Flexible (Dinámico).** Schemaless. Se p cambiar sobre la marcha.
Escalabilidad	Principalmente **vertical** (más potencia en un solo servidor). Costosa y con límites.	Principalmente **horizontal** (más servido Económica y casi ilimitada por diseño.
Propiedades	**ACID** (Atomicidad, Consistencia, Aislamiento, Durabilidad). Garantías fuertes.	**BASE** (Basically Available, Soft state, Ev consistency). Rendimiento y disponibili por encima de consistencia inmediata.
Lenguaje de Consulta	**SQL** (Structured Query Language). Estándar y muy potente para consultas complejas.	**No estándar.** Cada base de datos tiene propia API o lenguaje de consulta (ej. MongoDB Query Language, Cypher para grafos).

Característica	**Bases de Datos SQL (Relacionales)**	**Bases de Datos NoSQL**
Casos de Uso Típicos	Sistemas transaccionales (banca, ERP), donde la integridad es crítica.	Big Data, tiempo real, catálogos variable redes sociales, IoT.

Cuándo elegir SQL:

- Cuando la integridad y consistencia de los datos son críticas (ej. transacciones bancarias).
- Cuando la estructura de los datos es estable y predecible.
- Cuando se necesitan consultas complejas que relacionen muchas tablas.

Cuándo elegir NoSQL:

- Cuando se manejan enormes volúmenes de datos (Big Data).
- Cuando los datos no tienen una estructura fija o esta cambia constantemente.
- Cuando se necesita una escalabilidad masiva y económica.
- Cuando prima la velocidad de desarrollo y la flexibilidad.

Página 10: Ventajas, Desventajas y Futuro de NoSQL

Ventajas Principales

- **Escalabilidad Horizontal:** Capacidad casi ilimitada para crecer añadiendo servidores comunes.
- **Alto Rendimiento:** Optimizadas para operaciones simples de lectura/escritura a gran escala.
- **Flexibilidad del Esquema:** Permiten evolucionar la estructura de datos sin tiempos de inactividad.
- **Disponibilidad:** Diseñadas para seguir funcionando incluso con fallos en algunos nodos (gracias al Teorema CAP).

- **Modelos Especializados:** Ofrecen modelos de datos que se ajustan perfectamente a problemas específicos (grafos para relaciones complejas, clave-valor para cachés).

Desventajas y Desafíos

- **Consistencia Eventual:** La mayoría de sistemas NoSQL no garantizan consistencia inmediata, lo que puede ser un problema para ciertas aplicaciones.
- **Madurez:** Muchas tecnologías NoSQL son más jóvenes que los sistemas SQL, con menos herramientas y experiencia acumulada.
- **Falta de Estandarización:** Cada base de datos tiene su propio lenguaje y formas de trabajar, lo que dificulta el cambio de proveedor.
- **Soporte Limitado para Consultas Complejas:** No son ideales para consultas ad-hoc que involucren múltiples relaciones complejas (ahí ganan los grafos, pero no los documentales o clave-valor).

El Futuro: Políglota Persistente y Multimodelo

La tendencia actual no es hacia una tecnología que reemplace a todas las demás, sino hacia la **persistencia políglota**: utilizar la base de datos más adecuada para cada necesidad dentro de una misma aplicación (ej. PostgreSQL para transacciones, Redis para caché, Neo4j para recomendaciones). También están ganando popularidad las bases de datos **multimodelo**, que soportan varios paradigmas (documentos, grafos, clave-valor) en un solo motor, como ArangoDB o Azure Cosmos DB.